A Clustering Algorithm to Discover Low and High Density Hyper-Rectangles in Subspaces of Multidimensional Data

Carlos Ordonez Edward Omiecinski Shamkant Navathe Norberto Ezquerra *
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract

This paper presents a clustering algorithm to discover low and high density regions in subspaces of multi-dimensional data for Data Mining applications. High density regions generally refer to typical cases, whereas low density regions indicate infrequent and thus rare cases. For typical applications there is a large number of low density regions and a few of these are interesting. Regions are considered interesting when they have a minimum "volume" and involve some maximum number of dimensions. Our algorithm discovers high density regions (clusters) and low density regions (outliers, negative clusters, holes, empty regions) at the same time. In particular, our algorithm can find empty regions; that is, regions having no data points. The proposed algorithm is fast and simple. There is a large variety of applications in medicine, marketing, astronomy, finance, etc, where interesting and exceptional cases correspond to the low and high density regions discovered by our algorithm.

1 Introduction

1.1 Clustering

Data mining is defined as the automated discovery of previously unknown, nontrivial and potentially useful information from large databases [8]. Data mining is a combination of techniques taken from statistics, AI and databases [7]. There are several popular approaches used in data mining: association rules [3], classifier trees [7] and clustering [17] are some important ones. Clustering algorithms identify those regions that are more densely populated than others in multidimensional databases. This work presents a new clustering strategy designed to find interesting low and high density regions in subspaces of multidimensional data.

Clustering is a well researched problem in statistics [5, 11]. However, the proposed statistical approaches do not work well with databases because such schemes do not consider memory limitations and do not account for large data sets. Most of the work on clustering in the database community attempts to make clustering algorithms linear in the database size and at the same time minimize disk access. BIRCH [17] represents an important precursor in efficient clustering for databases. It is linear in database size and the number of passes is determined by a user-supplied accuracy. CLARANS [12] and DBSCAN [6] are also important clustering algorithms that work on spatial data. CLARANS uses randomized search and represents clusters by their medioids (most central point). DBSCAN clusters data points in dense regions separated by low density regions. One of the most important recent clustering algorithms is CLIQUE [2], that can discover clusters in subspaces of multidimensional data and which

*This work was supported in part by grant LM 06726-02 from the National Library of Medicine

exhibits several advantages over other clustering algorithms. We decided to follow a similar approach to CLIQUE. There is recent work on the problem of selecting subsets of dimensions being relevant to all clusters; this problem is called the projected clustering problem and the proposed algorithm is called PROCLUS [1]. This approach is specially useful to analyze sparse high dimensional data focusing on a few dimensions.

In general clustering algorithms partition the data set into several disjoint groups such that two points in the same group are similar and points accross groups are different according to some similarity criteria. There are two basic approaches to perform clustering: based on distance and based on density. Distance-based approaches identify those regions in which points are close to each other according to some distance function. On the other hand, density-based clustering finds those regions which are more highly populated than adjacent regions. Clustering algorithms can work in a top-down (hierarchical [11]) or a bottom-up (agglomerative) fashion. Bottom-up algorithms tend to be more accurate but slower.

CLIQUE exhibits several advantages over other clustering algorithms. It is fast, it can find clusters in subspaces of data, it is not sensitive to the order of input data points and its output is easy to understand. However, it has some disadvantages, and these were inherited or tackled by our algorithm. Some of its disadvantages include not having a systematic way to tune clustering parameters (to be explained later), several passes are required over the database, performance degrades with high dimensional dense regions (both from the number of passes required and the number of clusters generated), and inability to discover interesting low density regions. This algorithm is density-based and works in a bottom-up fashion increasing dimensionality for each pass it makes over the database. Its candidate cluster generation stage is very similar to the A-priori algorithm used to compute association rules described in [3].

High density regions are also called clusters in the context of this work. Some terms that are used to refer to low density regions in the literature are: negative clusters and holes in data [9]; BIRCH refers to unclustered data as outliers [17], whereas in [2] they are called non-dense and sparse. Not all sparse, unclustered or non-dense regions are interesting as we shall see and what we mean by low density is similar, but not equivalent. It is important to note that the algorithm proposed in [9] concentrates on finding holes only, but shows there has been a research interest in these regions.

High dimensional data represents a challenge for clustering algorithms. Each of them tackles the problem in a different way. In this work we attempt to find clusters and low density regions in *every* subspace at the expense of sacrificing dimensionality. Our clustering algorithm combines ideas taken from our previous work on association rules [14] and strong negative association rules [15].

The main points of our algorithm include the following:

- We use two density thresholds to discover low and high density regions. We find both low and high density regions in every subspace up to a maximum dimensionality. With these two parameters the user can get a better summarized picture of the data and can perform clustering in a more systematic way.

- We introduce a notion of volume to define the interestingness of results. High volume low density regions and low volume high density regions should be more interesting than their counterparts.

- We find low dimensionality regions with the minimum number of non-redundant dimensions required to meet the low density threshold. High dimensional low density regions are not interesting. Our algorithm, in particular, can find empty regions, that is, regions with zero points.

- We focus on hyper-rectangular regions instead of finding irregular regions described by a long disjunction of conjunctions (Disjunctive Normal Form used in [2]). This sacrifices the precision of region description but makes the program faster and its output easier to understand. We believe data mining is an exploratory tool and not a substitute for statistical analysis.

1.2 Motivation

The user may not be aware that his data has very few or no points in certain regions. Here we describe some situations in which these low density regions can be useful. For instance, home insurance claims may increase with bad weather during some month; but it can also be interesting to learn that in a particular month the number of claims was extremely low or perhaps zero, why?. Also consider the problem of detecting fraudulent transactions: take for instance a bank which is supposed to receive many deposits per week from many private companies but then the data miner for the bank finds that there is one month (4 weeks) when the number of deposits was extremely low. Finding empty regions in space could be interesting to astronomers. In astronomy dense groups of stars are called clusters. Large regions with few stars can be interesting when they separate many clusters. Empty regions are more interesting when they are large or when they are hidden inside a dense area. If those regions are found in projections of astronomy data they are more interesting when they involve a few variables.

We are interested in finding infrequent combinations of values. The user may even discover clues to cluster his data in a better way by looking at these unusual combinations. He can discover errors and missing information. Empty regions in space indicate infrequent or non-existent patterns. The user may expect to see some combination of attribute values happening together, but what if that doesn't happen?. That may be a hint to unexpected knowledge.

In general, for high dimensional data, large domains and many points we may expect to have an extremely high number of low density regions. Our algorithm tries to find those regions which we consider to be interesting. Those regions should involve a few dimensions, and have a relatively high volume, Consider for instance age in a table containing people data. Reporting very low density regions in which the longest empty intervals are only 3 months is probably not interesting; but if there were no people in a 10 year period (say from age 23 to 33) that is probably very interesting!. This interestingness is controlled by setting some clustering parameters to be explained later.

Paper outline. Section 2 presents our multidimensional model, definitions and a small example illustrating them. Section 3 presents our algorithm to find both low density (in particular empty) regions and high density regions (clusters). Section 3 also presents an example illustrating how the algorithm works. Performance experiments are described in Section 4.

2 Model and definitions

2.1 Multidimensional hyper-rectangles

In this section we provide definitions to be used throughout the paper. We give our definitions in the context of multidimensional data analysis.

Multidimensional space and subspace: Let $S = D_1 \times D_2 \ldots \times D_p$ be a p-dimensional space where each dimension D_i is a numerical domain. Any space whose dimensions are a subset of $\{D_1, D_2, \ldots, D_p\}$ is called a *subspace* of S. Each dimension D_i has a lower bound l_{D_i} and an upper bound u_{D_i}. S is then a a p-dimensional big hyper-rectangle whose sides are each dimension D_i.

Multidimensional database. Let M be a multidimensional database defined over space S. M consists of n points $x_1, x_2, \ldots, x_n$, each having p coordinates according to the dimensions of S.

This model can be extended to work on non-numeric data. Categorical dimensions can be converted into numerical dimensions by indexing their values; values $v_1, v_2, \ldots, v_j$ are converted into $1, 2, \ldots, j$. To find low density regions consecutive integers can be used to form large hyper-rectangles. For high density regions a hierarchy can be used to group them. Time can be treated as a numerical domain in which adequate time units are determined by the user; for instance the integers $1, 2, \ldots, 12$ can be the twelve months of one year. Clustering text data needs further research.

Interval and partition size σ: Each dimension D_i is *divided* into σ unitary intervals; σ will be an input parameter for our algorithm and we will call it partition size. An interval is denoted by $D_i : [l, u)$ where $l < u$. Each interval is left-closed and right-open and has length (according to D_i) $\frac{u_{D_i} - l_{D_i}}{\sigma}$. Based on this we define a special length measure to be used in the paper as follows. Given an interval its $length(D_i : [l, u))$ is the number of unitary intervals it overlaps. So the $length$ of each of the σ unitary intervals is 1 and the $length$ of the longest possible interval is σ.

Hyper-rectangle: With the definition of intervals we can now define a hyper-rectangle hr that will be the basic working unit for our algorithm. A hyper-rectangle on k distinct dimensions $D_1, D_2, \ldots D_k$, $1 \leq k \leq p$ is the region in k-dimensional space delimited by k intervals $hr = D_1 : [l_1, u_1), D_2 : [l_2, u_2], \ldots, D_k : [l_k, u_k)$. Such region is the intersection in space of all these k intervals. The intervals are also called sides. To make the paper more understandable all dimensions must appear in the same lexicographical order as they appear on S. Since we focus on rectangular regions we use the term region and hyper-rectangle equivalently.

Projection and extension: Let hr be a k-dimensional hyper-rectangle whose set of intervals (sides) is $I_{hr} = \{D_1 : [l_1, u_1), D_2 : [l_2, u_2], \ldots, D_k : [l_k, u_k)\}$. Let hr' be a k'-dimensional hyper-rectangle s.t. $k' < k$ and its set of intervals $I_{hr'} \subseteq I_{hr}$. We say that hr' is a *projection* of hr and hr is an *extension* of hr'. This allows us to manage subspaces.

Points contained in hyper-rectangles: Given a point $x_i \in M$ and a k-dimensional hyper-rectangle hr we say $x_i \in hr$ if $l_j \leq x'_{ij} < u_j$, where x' is the projection of x onto the k corresponding dimensions.

Volume: We now introduce a concept of *volume* (or hyper-volume). The volume of a k-dimensional hyper-rectangle hr is the product of the lengths of its sides, i.e. $vol(hr) = \Pi_{i=1}^{k} length([l_i, u_i))$. A hyper-rectangle whose sides are unitary intervals is also called unitary and then has volume 1. A hyper-rectangle hr s.t. $vol(hr) > 1$ is called "big". We also refer to unitary hyper-rectangles as "small".

Density: Since our clustering algorithm follows a density-based approach we define the density $density(hr)$ of hyper-rectangle hr to be the fraction of the n points from M contained in hr: i.e. $density(hr) = \frac{count(hr)}{n}$, where $count(hr)$ is the number of points contained in hr.

2.2 Input parameters for the algorithm

Number of unitary intervals per dimension: As mentioned before this parameter is called σ.

Density thresholds: We consider two density thresholds ϕ_1 and ϕ_2 s.t. $0 \leq \phi_2 < \phi_1$. These thresholds classify hyper-rectangles into L =low, M =medium and H =high density as follows. If $density(hr) \geq \phi_1 vol(hr)$ then $type(hr) = H$. If $density(hr) \leq \phi_2 vol(hr)$ then $type(hr) = L$. If $\phi_2 vol(hr) < density(hr) < \phi_1 vol(hr)$ then $type(hr) = M$. Note that these expressions are simpler for unitary hyper-rectangles. The volume factor scales density for big hyper-rectangles. ϕ_1 and ϕ_2 must not be close to each other; if $\phi_1 = \phi_2$ our approach reduces to having only dense and non-dense regions. Medium type (M) hyper-rectangles would generally not be considered interesting. In the example described in Section 3 we refer to density frequencies meaning $\phi_i n$ or $count(hr)$ for some hyper-rectangle hr.

Low density regions can have zero points; in this case the region is called empty. Equivalent terms for low density regions used in other works include negative clusters and holes [9]. Holes and empty regions can be used as equivalent terms. High density region and cluster are also equivalent. We consider both L and M hyper-rectangles to be sparse (non-dense) regions.

Dimensionality: We limit the maximum dimensionality of hyper-rectangles by a fourth parameter Δ. This parameter is critical to space consumption and algorithm speed because the total number of hyper-rectangles to be generated/analyzed increases as Δ increases. There is an interesting relationship between Δ and the density thresholds ϕ_1, ϕ_2 as we shall see in the next section.

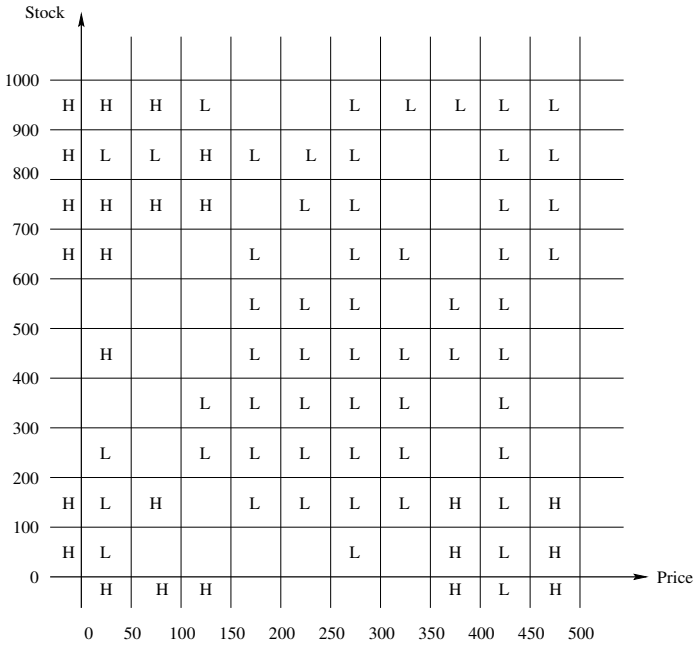

Figure 1. Hyper-rectangles for articles in a department store

2.3 Important remarks

We consider a hyper-rectangle of some type to be interesting if it has volume at least 1 and it is either H or L. The most interesting hyper-rectangles will be the big L hyper-rectangles (very sparse or empty) and small H hyper-rectangles (small dense regions). This idea conveys the concept of density.

Intuitively, we can imagine the entire space S to be a big multidimensional cube which is divided into σ^p small unitary cubes. And then the clustering algorithm will concentrate on selecting interesting projections of those cubes involving a few dimensions according to their density and their volume.

2.4 Example

In Figure 1 we show a simple example to illustrate the definitions given above. This example is about a department store which sells $n = 10,000$ articles. There are $p = 2$ dimensions considered for the articles: $Price$ and $Stock$. In the graph the X axis is the price of the articles which ranges from \$0 to \$500 and the Y axis represents the quantity available in stock ranging from 0 to a maximum of 1,000. Each dimension is partitioned into $\sigma = 10$ intervals. Taking the intersection in the plane of one interval from $Price$ and one interval from $Stock$ forms a unitary hyper-rectangle; which in the figure looks like a square; for instance $Price : [50, 100)$ $Stock : [800, 900)$ is one of them All these unitary hyper-rectangles form a grid. Each article will be a point that will fall into (belong to) one unitary hyper-rectangle.

The n articles are distributed all over the grid making some hyper-rectangles denser than others. The density thresholds are $\phi_1 = 0.04$ and $\phi_2 = 0.002$. Presumably, these values for ϕ_1,ϕ_2 would have been arrived at with some target application in mind. The unitary hyper-rectangles are classified according to their density into $L = low$ if their density is at most ϕ_2, $H = high$ if their density is at least ϕ_1 or $M = medium$ otherwise. So, in

those hyper-rectangles marked as L there are at most 20 articles, whereas in the H ones there are 400 articles or more. Adjacent unitary hyper-rectangles of the same type form bigger hyper-rectangles. M hyper-rectangles are blank to make the figure easier to understand.

In a typical department store inexpensive articles, in general, should have high quantities in stock. On the other hand, the stock of expensive articles should be low. In this case that is why some of the upper left hyper-rectangles are H, $Price$ in [0,100) and Stock in [900,1000), and some of the lower right are also H Price in [450,500) and $Stock$ less than 100. However, it is interesting that there are few articles whose $Price$ is in $[0, 100)$ and whose $Stock$ is in [800,900) (these hyper-rectangles are L); this may indicate that those articles are selling faster. Even further, the few cheap articles, whose $Price$ is in [0,50) and whose stock is in $[0, 300)$ may indicate very important articles for the store; those 3 L hyper-rectangles may point to "hot" selling articles; a fact that could be confirmed by looking at sales reports, but which could be surprising if the report says otherwise. Now, looking at expensive articles, it may be surprising that there are a few articles whose $Price$ is in $[400, 450)$ regardless of their stock; that is, the big L hyper-rectangle spanning 10 unitary hyper-rectangles. Finally for medium priced articles there is a big hole (region of L hyper-rectangles): $Price$ in [150,350) and $Stock$ in [100,500) and again, this may be unexpected.

The algorithm to be presented in the next section finds L and H hyper-rectangles involving a maximum number of dimensions. The hyper-rectangles labeled as M are considered to be of little interest and are not reported.

3 Discovering interesting L and H hyper-rectangles

3.1 Important theoretical results

The whole clustering process uses the following two lemmas to discover both L and H hyper-rectangles. Each of them is the counterpart of the other one for high density and low density hyper-rectangles respectively. Following the notation introduced in the previous section let S be a p-dimensional space and let hr be a k-dimensional hyper-rectangle, s.t. $k \leq p$ and $hr = D_1 : [l_1, u_1), D_2 : [l_2, u_2], \ldots, D_k : [l_k, u_k)$. In the proof text $\wedge =$ logical and.

- *Lemma 1:* If $type(hr) = H$ and hr' is a projection of hr then $type(hr') = H$.

  Proof: Suppose $type(hr) = H$ and hr' is a projection of hr and let $x \in hr$, some point. Then $density(hr) \geq \phi_1 vol(hr)$. Then since the hyper-rectangle is an intersection of intervals then $x \in D_1 : [l_1, u_1) \wedge x \in D_2 : [l_2, u_2) \wedge \ldots, \wedge x \in D_k : [l_k, u_k)$. Since hr' is a projection of hr then all its sides $D'_1 : [l'_1, u'_1), D_2 : [l'_2, u'_2], \ldots, D'_k : [l'_k, u'_k)$, are a subset of the sides of hr, $k' < k$. Then $x \in D'_1 : [l'_1, u'_1) \wedge x \in D'_2 : [l'_2, u'_2) \wedge \ldots, \wedge x \in D'_k : [l'_k, u'_k)$. Now since this holds $\forall x \in hr$ then the points contained in hr are a subset of the points in hr'. Therefore, $density(hr') \geq density(hr) \geq \phi_1 vol(hr)$. Since the sides of hr' are a subset of the sides of hr and each of them is a positive integer then $vol(hr) \geq vol(hr')$ (remember that vol is the product of side lengths). Being these quantities positive then $\phi_1 vol(hr) \geq \phi_1 vol(hr')$. Then $density(hr') \geq \phi_1 vol(hr')$, which proves the lemma.

- *Lemma 2:* If hr is a k-dimensional hyper-rectangle having $type(hr) = L$ and hr' is a k'-dimensional hyper-rectangle ($k' > k$) obtained by extending hr then $type(hr') = L$.

  Proof: Suppose $type(hr) = L$ and hr' is an extension of hr. There are two cases: that $hr' = \emptyset$ or that $hr' \neq \emptyset$. For the first case then $density(hr') = 0$ and then $density(hr') \leq \phi_2 vol(hr')$ then $type(hr') = L$, which proves the lemma.

  For the 2nd case ($hr' \neq \emptyset$) let $x \in hr'$ then $x \in hr$ since hr' is an extension of hr but if $y \in hr$ then it may be the case that $y \notin hr'$; therefore $density(hr) \geq density(hr') \geq 0$. Since hr' is an extension of hr and being vol a product of integers then $vol(hr') \geq vol(hr)$. By assumption $\phi_2 vol(hr) \geq density(hr)$ since

$type(hr) = L$ and by what we just proved $\phi_2 vol(hr') \geq \phi_2 vol(hr)$. Therefore, $\phi_2 vol(hr') \geq density(hr')$. So $type(hr') = L$, which concludes the proof.

It should be noted that both lemmas go in only one direction. For the first lemma if $type(hr') = H$ and hr' is a projection of hr, that is, we obtain hr' by extending hr with some intervals then $type(hr)$ may be any type out from H, M or even L because $density(hr) \leq density(hr')$. In the best case hr and hr' can have the same density but in general hr' is likely to contain more points.

Similarly, for the second lemma if hr' is an extension of hr nd $type(hr') = L$ then $type(hr)$ can also be L, M or H because $density(hr) \geq density(hr')$. Therefore, extending L hyper-rectangles by adding new intervals will not produce hyper-rectangles of any type but L. All the extensions will be of type L and at some point the density of some of them may become zero.

To understand the lemmas intuitevely we can go back to Figure 1. To understand lemma 1 we can look at the projections of every hyper-rectangle hr whose $type(hr) = H$: they are also H; the hyper-rectangle $Price : [50, 100), Stock : [700, 800)$ is one of them. On the other hand, looking at low density hyper-rectangles we can see that $Stock : [400, 450)$ is a 1-dimensional L hyper-rectangle having all extensions with dimension $Price$ being L. In this case extending $Stock : [400, 450)$ with $Price$ intervals would be redundant since each extension will be L according to lemma 2.

3.2 Algorithm

Here we present our algorithm. It has two major steps. The first step partitions the data space according to σ. The second step generates candidate hyper-rectangles according to the three other input parameters ϕ_1, ϕ_2, Δ and updates their densities to classify them. Each step is explained in further detail later.

- Input: M consisting of n points in p-dimensional space S and parameters $\sigma, \phi_1, \phi_2, \Delta$
- Output: L and H hyper-rectangles in subspaces of S.

Here is an outline of the algorithm:

1. Partition the p dimensions according to σ to obtain the intervals that will be the sides of hyper-rectangles. Consider all these intervals to be candidate sides of 1-dimensional hyper-rectangles
2. For $k = 1$ to Δ
   (a) Generate k-dimensional hyper-rectangles as candidates.
   (b) Make one pass over the database to compute the hyper-rectangles densities.
   (c) Classify the hyper-rectangles into types L, M and H according to ϕ_1 and ϕ_2.
   (d) Join adjacent hyper-rectangles of the same type to obtain bigger volume hyper-rectangles.
   (e) Report L and H hyper-rectangles of dimensionality k. L hyper-rectangles having small volume and high dimensionality can be filtered out. H hyper-rectangles that have low dimensionality and are projections of higher dimensional hyper-rectangles can also be eliminated.
   (f) Monitor the process. If results seen so far are not promising the whole process may be stopped to tune the input parameters and restart the algorithm.
   (g) Stop if all hyper-rectangles of dimensionality k are of type L, otherwise continue.

Here we describe important steps of the algorithm in further detail.

Input parameters: Δ is limited by available memory and CPU power. A very low ϕ_1 may render the algorithm slow as most hyper-rectangles will be H. A very high ϕ_2 will likely produce L hyper-rectangles. σ should be low enough to be able to group things, but high enough to have a fine grid to classify data. If σ is close to 1, say 2 or 3, then the algorithm may not distinguish between L and H hyper-rectangles. Our algorithm can find empty regions, i.e. regions with zero points by setting $\phi_2 = 0$; for some applications this may be useful. Ideally ϕ_1 and ϕ_2 should discriminate a lot of M hyper-rectangles.

Candidate generation: Hyper-rectangles are generated in a sorted fashion. They are sorted with respect to dimensions and intervals involved, i.e. given a hyper-rectangle hr as defined before $D_1 < D_2 <, \ldots, < D_k$, according to the order of dimensions that define S. And if $hr = D_1 : [l_1, u_1), D_2 : [l_2, u_2), \ldots, D_k : [l_k, u_k)$ is generated before $hr' = D'_1 : [l'_1, u'_1), D'_2 : [l'_2, u'_2), \ldots, D'_k : [l'_k, u'_k)$ then $D_i < D'_i$ or $D_i = D'_i \wedge u_i \leq l'_i$. The candidate generation procedure creates hyper-rectangles belonging to the same subspace consecutively. First, 1-dimensional subspaces are listed, then 2-dimensional spaces and so on. And within one dimension subspaces are listed according to the lexicographic order of the intervals involved. The candidate generation is optimized to consider only intervals in different dimensions; this makes the program faster. Note that H hyper-rectangles are always expanded up to Δ. The extension of H hyper-rectangles gives a clue at what dimensions uncluster the data.

Computing densities: Make one pass over the database to compute the hyper-rectangles densities. For each point $x_i \in M$ every hyper-rectangle $density(hr)$ is incremented if x_i falls into hr. Note that non-unitary hyper-rectangles densities are scaled wrt volume as defined. Since there is one pass for every dimensionality the algorithm becomes slow if Δ is high. The algorithm is not limited by database size since only a block is read at the time,

Hyper-rectangle classification: Classify the hyper-rectangles into types L, M and H according to ϕ_1 and ϕ_2. The intervals that are sides of M and H hyper-rectangles will participate in the next iteration to generate candidates. Intervals that are only sides of L hyper-rectangles are no longer used in future iterations.

Joining same type adjacent hyper-rectangles: Done to obtain bigger volume hyper-rectangles. We merge adjacent hyper-rectangles on each dimension. Use a depth-first strategy in this case; that is we take the first approximation as a good one. Then we create longer intervals belonging *only* to L and M hyper-rectangles. The intervals belonging to M hyper-rectangles will be the sides of higher volume and higher dimensionality L hyper-rectangles in future iterations; The big intervals that are sides of L or M hyper-rectangles overlap several (≥ 2) unitary intervals; all these overlapped unitary intervals are discarded. This has a *positive* impact on performance since that decreases the number of hyper-rectangles to be generated. Note that this is not done for intervals that are sides of H hyper-rectangles to preserve accuracy; that is, unitary H hyper-rectangles are used as the working unit for clusters.

Monitoring the process: If most, say more than 50%, hyper-rectangles are of type H then that may be an indication that ϕ_1 is too low. If most hyper-rectangles are of type L then that may warn that ϕ_2 is too high. Therefore, if results are not promising the whole process may be stopped to tune the input parameters and restart the algorithm.

Stopping criteria: All hyper-rectangles of dimensionality k are of type L. If at least one is of type M or L proceed to next dimensionality.

Report interesting hyper-rectangles: As we said before those are of type L and H. L hyper-rectangles having small volume and high dimensionality can be filtered out. H hyper-rectangles that have low dimensionality and are projections of higher dimensional hyper-rectangles can also be eliminated.

3.3 Implementation

The output of the program is limited to indicate the density, type and volume of each hyper-rectangle. We believe that this makes the output easy to understand. Both empty and dense hyper-rectangles are stored together

Point	1	2	3	4	5
1	1	4	7	1	2
2	1	2	3	5	6
3	0	4	8	8	2
4	1	4	9	1	2
5	1	1	3	2	4
6	0	5	8	0	9
7	0	2	3	5	5
8	3	1	3	3	0
9	5	9	6	2	1
10	6	8	6	3	9

Figure 2. Input file. $n = 10$, $p = 5$

in an array indexed by size and dimensions involved.

Only dimension info and hyper-rectangles of dimensionality k are stored in memory at a given time; so the memory space consumption is low. The input file is read one point at the time to update counts and thus we can process files of any size.

Dealing with the curse of dimensionality. Finding all clusters in all subspaces of high dimensional data is in general an intractable problem. Big hyper-rectangles in general reduce the number of passes over the input file since small hyper-rectangles (contained in the bigger ones) are discarded. Hyper-rectangles with longer sides and lower dimensionality may be more interesting than small hyper-rectangles with small sides but having high dimensionality. Such higher volume hyper-rectangles involving fewer attributes can be interpreted more easily by the user.

3.4 Related work

There has been an interest in finding empty regions before [9]. In this work the authors find large regions with no points in the full dimensional space; these regions are called holes in data. An important difference is that these regions are delimited with points and must have a minimum size in order to be reported. Some of the important points in which our approach is better are the following. The ability to discover almost empty regions and clusters concurrently. Our algorithm is faster for large sets of data. Our algorithm scales up better for high dimensional data. We find holes in subspaces of the data.

The other most important related work is [2] as we mentioned before. Some important differences include the following. We consider both high and low density regions in subspaces to be interesting. We do not prune out subspaces, but limit maximum dimensionality of regions instead. We use two density thresholds instead of one. We grow intervals to obtain longer intervals to be used in future iterations of the algorithm, and not only unitary ones. We use volume as a measure of interestingness. Our candidate generation is fast since we deal with longer intervals sooner and also intervals in the same dimension are not combined to generate higher dimensional hyper-rectangles. The process may be monitored to make clustering more interactive. Output is generated for each dimensionality; k-dimensional subspaces are reported after their corresponding hyper-rectangles have been classified. Our output is simpler, but less accurate in terms of cluster and low density region description.

3.5 Example

In Figure 2 we show a database with $n = 10$ records and $p = 5$ dimensions. The range for each dimension is $[0, 10)$. This database is the input for our algorithm implementation. It should be noted that we ran our program

Dimensionality	Count	Type	Vol	Intervals
1	7	H	1	1:[0,2)
1	3	L	4	1:[2,10)
1	4	H	1	2:[4,6)
1	0	L	1	2:[6,8)
1	4	H	1	3:[2,4)
1	3	H	1	5:[2,4)
2	4	H	1	1:[0,2) 2:[4,6)
2	0	L	1	1:[0,2) 2:[8,10)
2	3	H	1	1:[0,2) 3:[8,10)
2	0	L	4	2:[0,4) 3:[6,10)
2	4	L	6	2:[0,4) 4:[0,6)
2	3	L	6	2:[0,4) 5:[0,6)
2	0	L	2	2:[4,6) 4:[2,6)
2	1	L	3	2:[8,10) 5:[0,6)
3	3	H	1	1:[0,2) 2:[4,6) 3:[8,10)
3	2	L	6	1:[0,2) 2:[0,4) 5:[0,6)
3	1	L	1	3:[8,10) 4:[0,2) 5:[8,10)

Figure 3. L **and** H **hyper-rectangles.** $\sigma = 5, \phi_1 = 0.3, \phi_2 = 0.1, \Delta = 3$

with a small table in order to help the reader understand our approach. In the next section there are experimental results with files having a large number records and several dimensionalities. The $p = 5$ dimensions are partitioned into $\sigma = 5$ intervals. The corresponding frequency density thresholds for ϕ_1, ϕ_2 are 3 and 1 respectively. A part of the actual output of our program is shown on Figure 3. Some interesting hyper-rectangles are listed for this toy database. Note that some hyper-rectangles belonging to the same subspace are listed together.

The hyper-rectangle $1 : [2, 10)$ is an L hyper-rectangle with big volume and low dimensionality. that intersecting this interval with another interval will still make it an L hyper-rectangle and therefore the program does not expand it. On the other hand, $2 : [6, 8)$ is another one of the 6 1-dimensional L hyper-rectangles. $1 : [0, 2)$ and $2 : [4, 6)$ are the highest density H hyper-rectangle and then it is natural that they participate in the 3-dimensional H hyper-rectngle; in fact, these two intervals participate in all 3-dimensional H hyper-rectangles, but we only show one.

As dimensionality grows the number of unitary (small) L hyper-rectangles increases and we only show a couple of 3-dimensional ones. Note that 1-dimensional L hyper-rectangles no longer appear in any other hyper-rectangles since their expansion will only lower their density or preserve it in the best case. It is also important to point that the two 3-dimensional L hyper-rectangles have intervals that participate in H or M 2-dimensional hyper-rectangles. M hyper-rectangles are not reported since we consider them uninteresting to the user but can be listed if needed.

4 Performance evaluation

4.1 Theoretical analysis

Let $HRDS$ be the data structure in which hyper-rectangles are stored . The total cost to process M according to the parameters required by the algorithm is:

$$total\_cost = cost(HRDS) + cost(M)$$

.

If hyper-rectangles for each dimensionality fit in memory the the disk cost to process M is $O(\Delta n)$

- *Lemma 3:* The time complexity of our algorithm is $O(p^{\Delta} n)$.

  Proof: Let n be the number of p-dimensional points used as input for the algorithm and $\sigma, \phi_1, \phi_2, \Delta$ the algorithm input parameters. Let N_k be the number of candidate hyper-rectangles of dimensionality k.

  Step 1 involves executing σp computations to partition the p dimensions and therefore for big n we can ignore it.

  Step 2 involves executing the *for* loop of steps (a)-(g) Δ times. Steps (a) (c) (d) are proportional to the total number of hyper-rectangles N_k only. Step (b) involves reading the n points every time densities are computed and for each point accesing each candidate hyper-rectangle. Then this step is the most critical to algorithm speed. Step (b) is executed $N_k n$ times.

  If there are hyper-rectangles of type M or H for every possible dimension combination then there can be $N_k = p(p-1)(p-2)\ldots(p-k+1)$ hyper-rectangles. For high dimensional data we can approximate this number by $N_k \approx (p-k)^k \leq p^k$. For $k = \Delta$ in the last iteration this expression becomes $N_k \leq p^{\Delta}$. Note that in this step parameters ϕ_1, ϕ_2 come into play. If their value is too low every hyper-rectangle can be classified as H or M respectively. ϕ_1 is more critical since it is used for unitary hyper-rectangles whereas ϕ_2 is aimed to detect big L hyper-rectangles..

  Since (b) is the step that takes the longest time we can bound the algorithm execution time by the expression we derived looking at the highest dimensional hyper-rectangles. Step (b) is executed Δ times and if every k-dimensional hyper-rectangle is of type H then every projection was considered in previous iterations, and then the total time is bounded (for small Δ values) by $(p-\Delta)^{\Delta} 2^{\Delta} \Delta n = O(p^{\Delta} n)$ when $\Delta << p$.

In words this lemma states that the algorithm is linear in database size n, but it takes polynomial time in dimensionality p. The fact that the time is polynomial in p is good but not practical when dimensionality is very high (say > 100), Δ is also high (>10) or ϕ_1 is very low. This lemma gives a bound for the worst case but in practice we expect many interval combinations to be of type L and then be pruned out soon (no further expansion). Intuitively this lemma says that if all Δ-dimensional hyper-rectangles are of type H or type M then performance will be bad, but that should be unlikely in practice.

4.2 Experimental results

Our experiments were run on a Sun multiprocessor computer (*forge.cc.gatech.edu*) having 4 Sparc Processors, each running at 125MHz. This computer has 128 Mb of main memory and a disk array with several gigabytes of available storage space. Our algorithm implementation was done in the C++ language.

To test the program we generated synthetic data. Each dimension is normally distributed over the range $0 \ldots 100$ and has mean 50. The variance was varied between 1 and 3.5 standard deviations. This distribution produced a few dense regions and several low density regions.

The input parameters for the algorithm were set as follows. Partition size $\sigma = 10$. The H density threshold was set $\phi_1 = 0.13$. The L density threshold was set $\phi_2 = 0.06$. These two density thresholds produced a high number of hyper-rectangles to make the testing conditions harder. The database size n was varied for the first set of experiments and dimensionality p was varied for the second set of experiments. The program was run 3 times and the measured times were averaged.

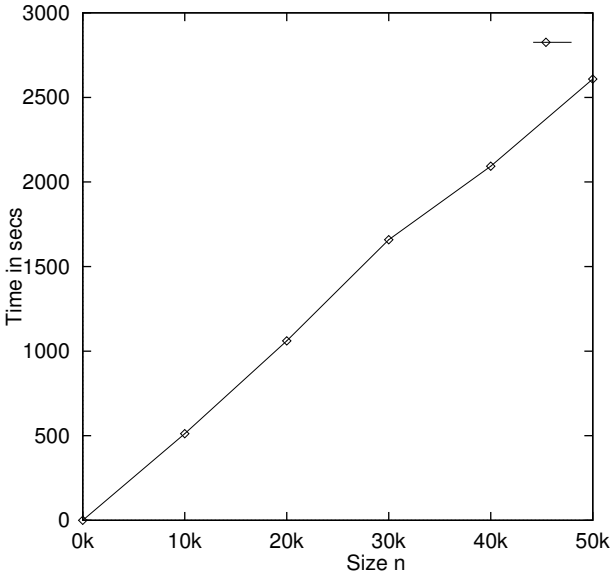

Figure 4. Performance for different database sizes n

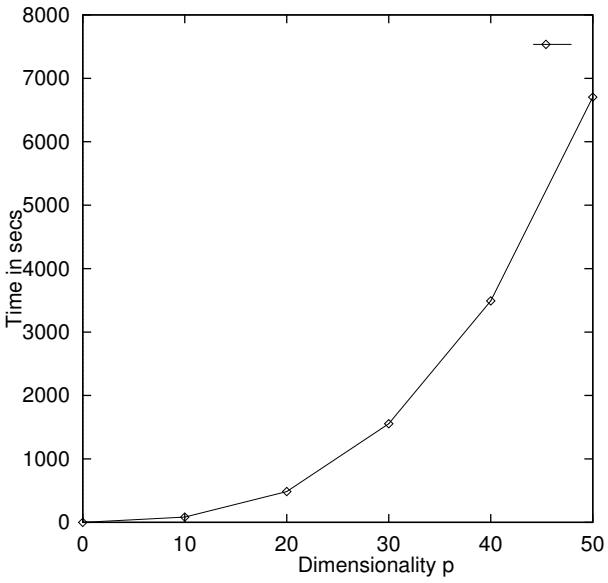

Figure 5. Performance for several dimensionalities p

Figure 4 presents experimental results with several databases of varying sizes but having the same dimensionality $p = 20$. As we can see the algorithm exhibits linear behavior. It should be noted that our implementation can handle databases of any size n. These small sizes were chosen to graphically show execution time.

Figure 5 presents times with databases of the same size $n = 10000$, but having several dimensionalities. The time is polynomial of degree $\Delta = 5$. As we can see performance degrades rapidly as dimensionality grows. We expect $\Delta \leq 5$ for high dimensional data, i.e. $p \geq 100$. For lower dimensionalities, say $p \leq 30$, we can set $\Delta \geq 8$

5 Conclusions

We presented a clustering algorithm that discovers low and high density regions in subspaces of multidimensional databases. The algorithm focuses on hyper-rectangular regions. Hyper-rectangles are classified into 3 parameterized types: low, medium and high density. Its output can be interpreted easily by the end user.

The algorithm scales well with database size and high dimensionality given tuned input parameters. There are 4 input parameters: σ, the number of intervals for partitioning, ϕ_1, ϕ_2, high and low density thresholds respectively, and Δ, a threshold for dimensionality. Discovered low density hyper-rectangles are not redundant in the sense that they involve a minimum number of intervals to meet ϕ_2.

We consider medium density hyper-rectangles and high dimensional low density regions to be of little interest. High and low density regions are considered important. The most interesting low density hyper-rectangles have low dimensionality and high volume. The most important high density hyper-rectangles should have low volume and a bit higher dimensionality than their counterpart.

Future research. We want to use this approach with medical data in a similar manner to our previous work on association rules used to help heart disease diagnosis [4] and to cluster images in image mining applications [13]. Our clustering algorithm is tuned to work with numerical data. So we would like to investigate how to grow hyper-rectangles involving categorical data using some taxonomy. Research on the problems associated with clustering time data. Investigate the possibility to use this appraoch to cluster text data. Efficiency can be improved by performing less passes using a similar approach to the association rule algorithm based on partitions [14] or finding high dimensional clusters sooner. More importantly, we want to extend this work to generate rules similar to Quantitative Association Rules [16], Distance-Based Association rules [10] and Strong negative association rules [15]. Since hyper-rectangles involve ranges these can be used to form rules.

It is our belief that both low and high density regions may be interesting for the data miner. Combined they give a more complete picture for data analysis. One of the purposes of data mining is discovering unexpected or surprising knowledge. Big low density regions may be surprising and thus they can be a good complement for clusters.

Acknowledgements

The first author would like to thank NCR corporation for providing a place and computer equipment to complete this work.

References

[1] C. Aggarwal, C. Procopiuc, J. Wolf, P. Yu, and J.S. Park. Fast algorithms for projected clustering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, 1999.

[2] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopolos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, 1998.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, August 29-September 1 1994.

[4] David Cooke, Carlos Ordonez, Ernie Garcia, Edward Omiecinski, Elyzabeth Krawczynska, R. Folks, C. Santana, Levien de Braal, and Norberto Ezquerra. Data mining of large myocardial perfusion spect (mps) databases to improve diagnostic decision making. *Journal of Nuclear Medicine*, 40(5), 1999.

[5] R. Dubes and A.K. Jain. *Clustering Methodologies in Exploratory Data Analysis*, pages 10–35. Academic Press, New York, 1980.

[6] Martin Easter, Hans Peter Kriegel, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, Portland, Oregon, 1996.

[7] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, November 1996.

[8] Usama Fayyad and G. Piateski-Shapiro. *From Data Mining to Knowledge Discovery*. MIT Press, 1995.

[9] B. Liu, L. Ku, and W. Hsu. Discovering interesting holes in data. In *Proceedings of the ICJAI Conference*, 1997.

[10] Renee Miller and Y. Yang. Association rules over interval data. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Tucson, Arizona, 1997.

[11] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 1983.

[12] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. of the VLDB Conference*, Santiago, Chile, 1994.

[13] Carlos Ordonez and Edward Omiecinski. Discovering association rules based on image content. In *Proc. of the IEEE Advances in Digital Libraries Conference (ADL99)*, Baltimore, Maryland, May 1999.

[14] Ashok Savasere, Edward Omiecinski, and Sham Navathe. An efficient algorithm for mining association rules. In *Proceedings of the VLDB Conference*, pages 432 – 444, Zurich, Switzerland, September 1995.

[15] Ashok Savasere, Edward Omiecinski, and Sham Navathe. Mining for strong negative associations in a large database of customer transactions. In *Proc. of the IEEE Conference on Data Engineering (ICDE)*, 1998.

[16] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the ACM SIGMOD Conference*, Montreal, Canada, 1996.

[17] T. Zhang, R. Rmakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. of the ACM SIGMOD Conference*, Montreal, Canada, 1996.